

## 1 Memory Management

1.1 For each part, choose one or more of the following memory segments where the data could be located: **text**, **data**, **heap**, **stack**

a) Local variables

b) Global variables

c) Constants (constant variables or values)

d) Functions (i.e. Machine Instructions)

e) Results of Dynamic Memory Allocation (**malloc** or **calloc**)

f) String Literals

1.2 Write the code necessary to allocate memory **on the heap** in the following scenarios:

- (a) An array of **arr** of **k** integers
- (b) A string **str** of length **p**. Note that a string's length is defined by **strlen**
- (c) An **n** × **m** matrix **mat** of integers initialized to zero.
- (d) Deallocate all but the first 5 values in an integer array **arr**. (Assume **arr** has more than 5 values).

```
int *arr = ... ;
```

1.3 Compare the following two implementations of a function which duplicates a string. Is either implementation correct?

```
char *strdup1(char *s) {
    int n = strlen(s);
    char *new_str = malloc((n + 1) * sizeof(char));
    for (int i = 0; i < n; i++) new_str[i] = s[i];
    return new_str;
}
```

```
char *strdup2(char *s) {
    int n = strlen(s);
    char *new_str = calloc(n + 1, sizeof(char));
    for (int i = 0; i < n; i++) new_str[i] = s[i];
    return new_str;
}
```

## 2 Pass-by-Who?

**2.1** The following functions may contain logic or syntax errors. Find and correct them.

- (a) Returns the sum of all the elements in `summands`.

```
int sum(int *summands) {
    int sum = 0;
    for (int i = 0; i < sizeof(summands); i++)
        sum += *(summands + i);
    return sum;
}
```

- (b) Increments all of the letters in the `string` where `string` points to the beginning of an array of arbitrary byte length `n`. Does not modify any other parts of the array's memory. Hint: `n` is not necessarily equivalent to `strlen(string)`.

```
void increment(char *string, int n) {
    for (int i = 0; i < n; i++)
        *(string + i)++;
}
```

- (c) Overwrites an input string `src` with "61C is awesome!" if there's room. Does nothing if there is not. Assume that `length` correctly represents the length of `src`.

```
void cs61c(char *src, size_t length) {
    char *srcptr, replaceptr;
    char replacement[16] = "61C is awesome!";
    srcptr = src;
    replaceptr = replacement;
    if (length >= 16) {
        for (int i = 0; i < 16; i++)
            *srcptr++ = *replaceptr++;
    }
}
```

2.2 Implement the following functions so that they work as described.

- (a) Swap the value of two `ints`. *Remain swapped after returning from this function.* Hint: Our answer is around three lines long.

```
void swap(_____, _____) {
```

```
}
```

- (b) Return the number of bytes in a string. *Do not use `strlen`.* Hint: Our answer is around 5 lines long.

```
int mystrlen(_____) {
```

```
}
```

### 3 Endianness

3.1 Suppose we run the following code on a 32b architecture:

```
uint32_t nums[2] = {10, 20};
uint32_t *q = (uint32_t *) nums;
uint32_t **p = &q;
```

Find the values located in memory at the byte cells for both a Big Endian and a Little Endian system given that:

- the array `nums` starts at address `0x36432100`
- `p`'s address is `0x10000000`

| Little Endian |      | Big Endian |      |
|---------------|------|------------|------|
| 0xFFFFFFFF    | ...  | 0xFFFFFFFF | ...  |
| 0x36432107    |      | 0x36432107 |      |
|               |      |            |      |
|               |      |            |      |
|               |      |            |      |
|               |      |            |      |
|               |      |            |      |
|               |      |            |      |
| 0x36432100    |      | 0x36432100 |      |
|               | ...  |            | ...  |
| 0x20000003    |      | 0x20000003 |      |
|               |      |            |      |
|               |      |            |      |
| 0x20000000    |      | 0x20000000 |      |
|               | ...  |            | ...  |
| 0x10000003    | 0x20 | 0x10000003 | 0x00 |
|               | 0x00 |            | 0x00 |
|               | 0x00 |            | 0x00 |
| 0x10000000    | 0x00 | 0x10000000 | 0x20 |
|               | ...  |            | ...  |

3.2 Suppose we add an additional instruction (line #4) to the end of the previous code block:

```
uint32_t nums[2] = {10, 20};  
uint32_t *q = (uint32_t *) nums;  
uint32_t **p = &q;  
uint64_t *y = (uint64_t *) nums;
```

Provide answers for the following questions for both a Big Endian system and Little Endian system:

- 1) What does `*y` evaluate to?
- 2) What does `&q` evaluate to?
- 3) What does `&nums` evaluate to?
- 4) What does `*(q + 1)` evaluate to?

